```java
// return the shortest path from v to s as an Iterable
    public Iterable<String> pathTo(String v) {
        Stack<String> path = new Stack<String>();
        while (v != null && dist.contains(v)) {
            path.push(v);
            v = prev.get(v);
        }
        return path;
    }
```

Using a queue instead of a stack in the `pathTo()` method would result in reversing the order of the returned path.

Here's what would happen step by step:

1. The method starts at the destination `v` and works backwards to the source.
2. With a stack, each vertex is pushed, so when you iterate over the stack later, you get the path from source to destination.
3. With a queue, the first vertex added would be the destination, and the last would be the source.
4. When you iterate over the queue, you'd get the path backwards, from destination to source.

With a Stack:

- A stack follows **Last-In-First-Out (LIFO)** order, so when you `push()`, the vertices as you trace back from `v` to `s`, you push them onto the stack, constructing the stack from the destination to the source. This ensures that the first vertex traced (the destination) is on top of the stack, and the last vertex traced (the start) ends up at the bottom. When the stack is iterated (or "popped"), the path is returned in the correct order from start (`s`) to destination (`v`).

With a Queue:

- A queue follows **First-In-First-Out (FIFO)** order, so if you `add()` vertices as you trace back from `v` to `s`, the first vertex traced (the destination) will be at the front of the queue. The last vertex traced (the start) will be at the back. When you iterate over the queue, the path would be in reverse order, i.e., from `v` (the destination) to `s` (the start).

Example:

If the path from `s` to `v` is: `s -> a -> b -> v`

- With a **stack**, the method will first `push(v)`, then `push(b)`, `push(a)`, and finally `push(s)`. Iterating over the stack will return `s -> a -> b -> v`, which is the

correct order.

- With a **queue**, if you `add(v)`, then `add(b)`, `add(a)`, and finally `add(s)`, iterating over the queue will return `v -> b -> a -> s`, which is the **reverse** of the correct path.

Conclusion:

Using a queue instead of a stack would result in the returned path being in reverse order (from `v` to `s` instead of from `s` to `v`).